PROGRAMMING INTERFACE FOR LICENSING

FIELD OF THE INVENTION

[0001] The present invention relates generally to the field of computer software, and, more particularly, to a programming interface that supports the enforcement of electronic licenses.

BACKGROUND OF THE INVENTION

[0002] Commercially-produced software has traditionally been made available under a license that defines the permissible terms of the software's use. When the practice of software licensing first began, the license generally took the form of a legal document that defined the user's rights with respect to the software. Such a document relied upon the legal system for enforcement. It has since become desirable for licenses to be enforced electronically – i.e., it is desirable that a computer program contain code that actively discourages or prevents use of the software in a manner that is contrary to the license.

[0003] Most software that provides for electronic license enforcement provides its own infrastructure to manage the licensing of the software and the use of the licenses. Thus, a typical commercial software product may include not only the code to perform the product's core function, but may also carry with it the code to obtain, evaluate, protect, and manage licenses for the

software. For each software vendor to develop and incorporate this infrastructure into its software is often a wasteful duplication of effort. It is therefore desirable to provide a system that performs the basic functions related to software licensing, where the system can be used by a broad variety of software applications in a uniform and defined way.

[0004] In view of the foregoing, there is a need for a mechanism that overcomes the drawbacks of the prior art.

SUMMARY OF THE INVENTION

[0005] The present invention provides a software licensing Application Programming Interface (API) that provides certain licensing functions for use by software products. A license service performs functions relating to the use of licenses, and exposes these functions to software products through the API. The service performs functions such as obtaining licenses, storing and managing licenses, protecting licenses from tampering, evaluating a license's validity, and evaluating whether a license is correctly bound to the machine and/or software product on which it is used. The software is able to make use of this functionality by calling the methods of the API.

[0006] In a typical use of the API, a software product calls an "open" API method in order to obtain a unique handle that is used by the license service to identify the application. The software product then calls a "consume right" API method. "Consume," in this context, means the exercise of a specified right. The call to the "consume right" method is parameterized by the software product's handle, and by the name of the right to be consumed. The license service then attempts to locate one or more valid, correctly bound licenses that contains the named right. If no such license exists, then the software product is notified of the failure. If such licenses exist, then the right is bound to one of the licenses, and the calling software product is notified of the binding. In such a case, the software product knows that the right exists, and can perform whatever functions are associated with this right.

[0007] In a preferred embodiment, the license service does not define what the software can or cannot do under the right, or enforce substantive constraints on the use of the software. Rather, the license service manages the licenses in such a way that a software product can determine by calling the API whether a right does, or does not, exist, so that the software can behave accordingly. For example, a right may be called "run," indicating that the user has the right to run

the software product. The software product can use the API to determine whether there is a valid (and correctly bound, and non-expired) right to run the software. However, if the API call returns with an indication that there is no right to run the software, it is up to the software to cease operation or take some other action based on the non-existence of this right.

[0008] A right may be associated with information, which becomes available after a successful call to the "consume right" method. For example, a given software product may have individual rules about when it is permissible to edit, print, save, etc., and these rules can be stored in the license that contains the right. The API provides a "get information" method that allows this information to be retrieved from the license.

[0009] Other features of the invention are described below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The foregoing summary, as well as the following detailed description of preferred embodiments, is better understood when read in conjunction with the appended drawings. For the purpose of illustrating the invention, there is shown in the drawings exemplary constructions of the invention; however, the invention is not limited to the specific methods and instrumentalities disclosed. In the drawings:

[0011] FIG. 1 is a block diagram of an example computing environment in which aspects of the invention may be implemented;

[0012] FIG. 2 is a block diagram of an architecture in which a system performs licensing functions and exposes an API for use by software products;

[0013] FIG. 3 is a flow diagram of a method through which a software product uses a licensing API;

[0014] FIG. 4 is a flow diagram of a method by which a software product consumes a right; and

[0015] FIG. 5 is a flow diagram of a method by which a software product retrieves information relating to a consumed right.

DETAILED DESCRIPTION OF THE INVENTION

Overview

[0016] The use of commercial software is typically governed by a license, and it has become increasingly common for this license to be embodied in an electronic form that can be enforced by the software itself. One challenge in creating an electronic licensing system is that an infrastructure is needed to manage the use of the licenses. Replicating this infrastructure for every software product is cumbersome and wasteful. The present invention provides an API that allows different software products to use a common infrastructure that performs various licensing functions.

Exemplary Computing Arrangement

[0017] FIG. 1 shows an exemplary computing environment in which aspects of the invention may be implemented. The computing system environment 100 is only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computing environment 100 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated in the exemplary operating environment 100.

[0018] The invention is operational with numerous other general purpose or special purpose computing system environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, embedded systems, distributed computing environments that include any of the above systems or devices, and the like.

[0019] The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network or other data transmission medium. In a distributed computing

environment, program modules and other data may be located in both local and remote computer storage media including memory storage devices.

[0020] With reference to FIG. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a computer 110. Components of computer 110 may include, but are not limited to, a processing unit 120, a system memory 130, and a system bus 121 that couples various system components including the system memory to the processing unit 120. The processing unit 120 may represent multiple logical processing units such as those supported on a multi-threaded processor. The system bus 121 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus (also known as Mezzanine bus). The system bus 121 may also be implemented as a point-to-point connection, switching fabric, or the like, among the communicating devices.

[0021] Computer 110 typically includes a variety of computer readable media. Computer readable media can be any available media that can be accessed by computer 110 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable media may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CDROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can accessed by computer 110. Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode

information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, RF, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

[0022] The system memory 130 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 131 and random access memory (RAM) 132. A basic input/output system 133 (BIOS), containing the basic routines that help to transfer information between elements within computer 110, such as during start-up, is typically stored in ROM 131. RAM 132 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 120. By way of example, and not limitation, FIG. 1 illustrates operating system 134, application programs 135, other program modules 136, and program data 137.

[0023] The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, FIG. 1 illustrates a hard disk drive 140 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156, such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

[0024] The drives and their associated computer storage media discussed above and illustrated in FIG. 1, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In FIG. 1, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data

137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 20 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse, trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device is also connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through an output peripheral interface 195

[0025] The computer 110 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 180. The remote computer 180 may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 110, although only a memory storage device 181 has been illustrated in FIG. 1. The logical connections depicted in FIG. 1 include a local area network (LAN) 171 and a wide area network (WAN) 173, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[0026] When used in a LAN networking environment, the computer 110 is connected to the LAN 171 through a network interface or adapter 170. When used in a WAN networking environment, the computer 110 typically includes a modem 172 or other means for establishing communications over the WAN 173, such as the Internet. The modem 172, which may be internal or external, may be connected to the system bus 121 via the user input interface 160, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 110, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 1 illustrates remote application programs 185 as residing on memory device 181. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

Software Licensing Service

[0027] FIG. 2 shows an example system that provides a software licensing service 202. Software licensing service 202 operates inside of computer 110 (shown in FIG. 1). In one example, software licensing service 202 is part of an operating system that executes on computer 110. Software licensing service maintains a license store 204 in which license files for software are stored. License files may, for example, be eXtensible Rights Markup Language (XrML) files that specify rights to software, and that also may specify various types of conditions on the exercise of those rights. Software 204 also maintains a trust store 206. Trust store 206 stores un-authenticatable, dynamic data in a tamper-resistant manner; trust store 206 stores data that is used in the license validation process. For example, certain licenses may have expiration dates, and, in order to prevent the expiration data from being circumvented through clock rollback, the current time (and elapsed time) may be periodically stored in trust store 206 to ensure that the clock is always moving forward.

[0028] Software licensing service 202 manages license store 204 and trust store 206, and also performs various functions relating to the licensing of software. For example, software licensing service 202 may contain modules that parse license files, modules that enforce the binding of a license to a particular machine and/or to a particular instance of a software product, and a secure timer/counter (that uses trust store 206 in the manner described above).

[0029] Software licensing service 202 exposes an application programming interface (API) 208 that allows application software (such as application 135) to use software licensing service 202. Application 135 may invoke the features of software licensing service 208 by making local procedure call (LPC) to the methods of API 208. An example set of API methods that may be exposed by software licensing service 202 is described below.

[0030] The manner in which API 208 is used by an application is described with reference to FIG. 3. Initially, the application makes an API call (302). Service 202 then processes the API call (304), and returns the result of the API call to the application 306. For example, an API call may request to exercise ("consume") a right granted in a license, or to retrieve information from a license. The application then receives the result of the API call, and determines, based on that result, what the application's behavior should be (308). In other words, software licensing service 202, in a

preferred embodiment, does not enforce a license directly, but rather provides the infrastructure through which licenses can be managed and used. For example, if an application makes an API call to consume a right and service 202 determines that there is no valid license granting this right, service 202, in a preferred embodiment, does not prevent the application from running, but instead informs the application that the right is not available. Thus, the application can use its own mechanisms to determine what to do in response to the unavailability of the right. This facet of the API provides software vendors the flexibility to decide how the licensing infrastructure provided by service 202 should be used. In another implementation, application can be bound to licensing service functionality.

Example Software Licensing API

[0031] The following is an example set of API methods that may be exposed by a software licensing service:

SLOpen

[0032] The SLOpen function opens a SL client context handle that must be used for all subsequent SL API calls. (Throughout the example API descriptions, "SL" shall refer to the software licensing service. Thus, the SL client context handle is the handle used by a client in communicating with the software licensing service.)

HRESULT
SLOpen(
CONST GUID* *pguidApp,*
HSLC* *phSLC*
);

Parameters

pguidApp

[0033] [in] Pointer to application GUID that uniquely identifies an application. If this argument is NULL, an E_INVALIDARG error is returned.

phSLC

[0034] [out] SL client context Handle or INVALID_HANDLE_VALUE if failed.

Remarks

[0035] Use the SLClose function to close an context handle returned by SLOpen.

[0036] Application GUID: Unique ID of application. For the WINDOWS version of the MICROSOFT OFFICE application suite, WinWord has an Application GUID which is different from Excel's Application GUID. For Windows, Windows itself is an application, although it is a composite of many programs.

[0037] In Office case, A user can install both Office Suite and WinWord standalone products on the machine. From SL point of view, the WinWord in both products has same Application GUID. WinWord's Application GUID associated to two Product GUID. In other words, WinWord can use either Office Suite's product license or WinWord's product license.

[0038] When SLOpen succeeds:

[0039] A RPC (remote procedure call) binding has been established.

[0040] A context memory is created on SL service. The context is used to keep the status information for the caller of the client.

[0041] The SLC handle is like a file handle. A process can open multiple SL context handles but handles are valid within the caller process.

Returns

[0042] Success or failure

SLClose

[0043] The SLClose function closes an opened SL client context handle. Any information in the context is released automatically.

HRESULT
SLClose(
  HSLC *hSLC*
);

Parameters

hSLC

[0044] [in] Handle to current SL client context handle

Remarks

[0045] When SLClose is done, the RPC binding is released, and the context is destroyed.

Returns

[0046] Success or failure.

SLInstall

[0047] The SLInstall function installs applications' licenses and registers applications' information.

```
HRESULT SLInstall(
  HSLC              hSLC,
  CONST SL_PRODKEY* pAppPrdKey,
  DWORD             dwNumOfApps,
  CONST GUID*       pguidApps,
  DWORD             dwNumOfLicFiles,
  PCWSTR            ppszLicFiles[],
  BOOL              bVerify
);
```

Parameters

hSLC

[0048] [in] Handle to current SL client context handle

pAppPrdKey

[0049] [in] Application product key structure. The product key can be Microsoft product key format or Application's product key format.

```
typedef struct _tagSL_PRODKEY
{
```

```
    DWORD cbSize;                                   // Size of SL_PRODKEY
structure
    DWORD dwVersion;                // Version of SL_PRODKEY structure
    WCHAR szProdKey[MAX_PRODKEYSTR_SIZE+1];
    SL_PRODKEY_TYPE eProdKeyType;          // Type of Product key
    SL_CUSTOM_PRODKEY_INFO CustomPrdKeyInfo;   // Customer product key info
} SL_PRODKEY;
```

The eProdKeyType can be one of following values:

SL_PRODKEY_CUSTOM

SL_PRODKEY_MS2002

SL_PRODKEY_MS2003

[0050] If the Product Key type is non-MS Product Key (i.e. eProdKeyType == SL_PRODKEY_CUSTOM), the caller has to fill in its custom product key information. If the product uses MS Product Key, then CustomPrdKeyInfo can be ignored.

```
typedef struct _tagSL_CUSTOM_PRODKEY_INFO
    DWORD dwSKUID;                  // Unique ID for specific SKU, for example
Group ID in MS PID.
    DWORD dwSerialNumber;       // unique serial number, e.g. channel + sequence
number in MS PID.
} SL_CUSTOM_PRODKEY_INFO;
```

[0051] Current version number of SL_PRODKEY is 1. The caller can use SL_CURRENT_PRODKEY_VERSION in dwVersion field.

dwNumOfApps

[0052] [in] The number of application GUID in pguidApps.

pguidApps

[0053] [in] A list of application of GUID. The application GUID represents the application that the license is being installed for. For example, Office Setup program can call this function to install the license(s) for Word, Excel by specifying each application GUID in pguidApps. *pguidApp* cannot be NULL here.

dwNumOfApps

[0054] [in] The number of license files.

ppszLicFile

[0055] [in] File names in array of strings .

Returns

[0056] Success or failure

SLUninstall

[0057] The SLUninstall function uninstalls a product's license from an application.

HRESULT SLUninstall (
  HSLC *hSLC,*
  CONST SL_PRODKEY* *pAppPrdKey*
);

Parameters

hSLC

[0058] [in] Handle to current SL client context handle

pAppPrdKey

[0059] [in] See above defininition in connection with SLInstall.

Remarks

[0060] An application could have more than one product licenses. For example, when the user uninstalls Office suite, the association between Office Suite license and WinWord should be removed, but the license from WinWord Standalone product should not be removed.

[0061] When SLUninstall succeeds:

[0062] The information associated with this Product Key is removed. (see SLInstall, above, for the associated information)

[0063] The product keys associated with the Application GUID is removed.

[0064] The license files associated with the product GUID are preferably still kept.

Returns

[0065] Success or failure

SLConsumeRight

[0066] The SLConsumeRight function lets an application to examine or exercise the rights on a locally-stored license. Calling this function binds a license to the right mentioned in *pszRightName*. If this right cannot be exercised by the current caller, then the application fails. If the function succeeds, the action associated with the right can be executed (like decreasing usage count, decreasing time quota, or nothing)

HRESULT SLConsumeRight(
HSLC *hSLC,*
PCWSTR *pszRightName,*
SL_ASYNC_CONTEXT* *pAsyncContext*
);

Parameters

hSLC

[0067] [in] Handle to current SL client context handle

pszRightName

[0068] [in] The name of right needs to be evaluated. In current design, the right name is defined by applications. SL opens the license and evaluates the condition based on the right name.

pAsyncContext

[0069] [in/out] If *pAsyncContext* is NULL, this function works in synchronous mode, otherwise, the function works in asynchronous mode. **SL_ASYNC_CONTEXT** is opaque to caller and managed by SLC.

Remarks

[0070] All licenses associated with the application GUID (specified in SLOpen) will be conceptually combined in one logic license.

[0071] If their are multiple consumable grants of the right, then the license with higher priority will be consumed first.

Returns

[0072] Success or failure

SLInitializeAsyncContext

[0073] The SLInitializeAsyncContext function initializes the asynchronous context for SLC functions to make asynchronous call.

```
HRESULT SLInitializeAsyncContext(
  SL_ASYNC_CONTEXT* pAsyncContext,      // asynchronous context
  HANDLE hEvent,                        // event handle
  PVOID pvReserved                      // reserved, NULL
);
```

Parameters

pAsyncContext

[0074] [in/out] pointer to asynchronous context that contains asynchronous call information..

hEvent

[0075] [in] The event object used for synchronization.

pvReserved

[0076] [in] reserved for extension.

Returns

[0077] Success or failure.

SLCancelAsyncCall

[0078] The SLCancelAsyncCall function is used to cancel an asynchronous call.

HRESULT SLCancelAsyncCall(

SL_ASYNC_CONTEXT* *pAsyncContext,* // asynchronous context

BOOL *fAbortCall* // cancel immediately

};

Parameters

pAsyncContext

[0079] [in] asynchronous context for SL asynchronous call.

fAbortCall

[0080] [in] If TRUE, the call is cancelled immediately. If FALSE, wait for the SL to complete the call.

Remarks

[0081] There are two ways for the caller to request cancellation of an asynchronous call—abortive and nonabortive. In an abortive cancel (fAbortCall is TRUE), the SLCancelAsyncCall function sends a cancel notification to the SLC and the asynchronous call is canceled immediately, without waiting for a response from the SLC. In a nonabortive cancel (fAbortCall is FALSE) the SLCancelAsyncCall function notifies SLC of the cancel and the caller waits for SLC to complete the call.

Returns

[0082] Success or failure.

SLCompleteAsyncCall

[0083] The SLCompleteAsyncCall function is used to complete an SLC asynchronous call.

HRESULT SLCompleteAsyncCall(

SL_ASYNC_CONTEXT* *pAsyncContext,* // asynchronous context

HRESULT* *phrAsyncCall* // error code of the submitted asynchronous call

);

Parameters

pAsyncContext

[0084] [in] asynchronous context for SL asynchronous call.

phrAsyncCall

[0085] [out] the error code of the submitted asynchronous call.

Remarks

[0086] If the caller calls this function before the reply has arrived, the call returns E_SLC_ASYNC_CALL_PENDING. The buffer must be valid and it must be big enough to receive the return value. If the call does not return E_SLC_ASYNC_CALL_PENDING, this SLCompleteAsyncCall invocation is final for the asynchronous call. After this function call, regardless of success or failure, all resources allocated for this asynchronous call are freed. (Subsequent calls to the SLCompleteAsyncCall or SLCancelAsyncCall functions have undefined results until a new call on the SL_ASYNC_CONTEXT structure is initiated).

Returns

Value	**Meaning**

S_OK	The call was completed successfully.
E_SLC_INVALID_ASYNC_CONTEXT	The asynchronous call context is not valid.
E_SLC_ ASYNC_CALL_PENDING	The call has not yet completed.
E_SLC_CALL_CANCELLED	The call was cancelled.

SLGetInformation

[0087] The SLGetLicenseInfo function is used to get a variety of information.

```
HRESULT SLGetInfomation(
  HSLC      hSLC,        // SL client context handle
  DWORD     dwCategory,  // The category of information to retrieve
  PCWSTR    pszKeyName,  // Name of the Key
  DWORD*    pdwType,     // Type of value
  SIZE_T*   pcbValue,    // Size of value
  PBYTE*    ppbValue     // Pointer to buffer of value
);
```

Parameters

hSLC

[0088] [in] Handle to current SL client context handle

dwCategory

[0089] [in] The category of information.

Category	Meaning
SL_CAT_RIGHTDATA	Get the information from bound right. The license has to be consumed successfully before getting these right data.
SL_CAT_DYNAMICPROPERTY	Get the information that is not in the license but calculating in the run-time. For example,

RemainingGracePeriodDays. The right has to be consumed before calling.

<table>
<tr><th>Name</th><th>Meaning</th></tr>
<tr><td>RemainingGracePeriodDays : DWORD</td><td>The grace period is defined in out-of-box license. Once the application is installed, the time is counting down. Applications can check remaining grace period after they have consumed license.</td></tr>
<tr><td>ActivationStatus : DWORD</td><td>After applications consumed license, it can get consumed license type. The return value could be:
<table>
<tr><td>SL_LIC_OOB</td><td>The consumed license is out-of-box license.</td></tr>
<tr><td>SL_LIC_ACQUIRED</td><td>The consumed license is acquired license.</td></tr>
<tr><td>SL_LIC_NONE</td><td>No license is available.</td></tr>
</table></td></tr>
</table>

SL_CAT_SERVICEINFO — Get the information that is not dependent on license. The caller can get this category of information without consuming license.

Name	Meaning
SLVersion: DWORD	The version of SL. 1.2.3.4 format.
HWID:BINARY	Current HWID

SL_CAT_WINDOWSINFO — Get information that is bound right property in Windows license. This is for Componentization. Windows License has been consumed by SL service and SL service keeps the bound right properties.

SL_CAT_ENUMLICINFO — When SLEnumLicenseis called and is succeed, the caller can query the information of enumerated license by using this category.

pszKeyName

[0090] [in] the name of the key. E.g. BuildNumber

pdwType

[0091] [out] type of data

Value	Meaning
SL_DATATYPE_SZ	Unicode string
SL_DATATYPE_DWORD	DWORD
SL_DATATYPE_BINARY	Binary

pcbValue

[0092] [out] Size of the buffer allocated (in bytes).

ppbValue

[0093] [out] If successful, the data is returned in the buffer allocated by SLC. The caller has to call SLFreeMemory to free the memory.

Returns

[0094] Success or failure

SLAcquireLicense

[0095] The SLAcquireLicense function is used to acquire on-line license for the user. SLC enumerates the product keys associated with the Application and picks up the product key with highest product priority (see SLInstall, registration information). Then SL gets the clearing house URL from out-of-box license and connects to clearing house to get a license.

[0096] SLAcquireLicense could be a lengthy process. Applications can call this function in asynchronous mode by specifying *pAsyncContext (NULL means synchronous mode)*.

HRESULT SLAcquireLicense(

HSLC *hSLC,* // SL client context handle

PCWSTR *pszProdKeyHash,* // hash of product key

PCWSTR *pszPublishLicense,* // string of publishing license.

SL_ASYNC_CONTEXT* *pAsyncContext*

);

Parameters

hSLC

[0097] [in] Handle to current SL client context handle

pszProdKeyHash

[0098] [in] string of product key hash. The product key hash is created when SLInstall is called and is maintained by the licensing service.

pszPublishingLicense

[0099] [in] string of publishing license.

pAsyncContext

[0100] [in] asynchronous context for SL asynchronous call.

Remarks

[0101] The acquired license will be stored in license store accordingly and the license information will be registered, too. (see SLInstall)

[0102] Applications might need to add more client information to license. The application can put the information to pbAppData in the call and this data will be sent to clearing house.

[0103] When this function succeeds:

[0104] It sent necessary binding information to the specified license server.

[0105] It receives the license from license server.

[0106] It stored the license in license store. See description of SLInstall, above, for how the license file is stored.

Returns

[0107] Success or failure.

SLGenerateTextChallenge

[0108] Generates and installation challenge text to be routed to a license issuer in an out of band fashion (telephone, email, file share, etc).

HRESULT SLGenerateTextChallenge(

HSLC *hSLC,* // SL client context handle

PCWSTR *pszProdKeyHash,* // string of product key hash

BOOL *fSingleSession,* // Single session only?

PWSTR **ppszChallenge* // Pointer to buffer to hold challenge text

);

Parameters

hSLC

[0109] [in] Handle to current SL client context handle

pszProdKeyHash

[0110] [in] String of product key hash. The product key hash is created when SLInstall is called and maintained by the licensing service.

bSingleSession

[0111] [in] Specifies whether or not the corresponding text response from the license issue will be valid only for the lifetime of this SLC session handle.

ppszChallenge

[0112] [out] If successful, the text challenge is returned in the buffer allocated by SLC. The caller needs to call SLFreeMemory to free the allocated memory.

Returns

[0113] Success or failure.

SLDepositTextResponse

[0114] Deposits the response to an installation challenge text in the licensing system. Used to activate a license with a conditional access code. Only valid if there is an outstanding text challenge which has been issued for a license. If the original license specified that the challenge was only valid for a single session, this API must be called with the response before the SLC handle is closed or depositing the response will fail.

HRESULT SLDepositTextResponse(

HSLC *hSLC,* // SL client context handle
PCWSTR *pszProdKeyHash,*
PWSTR *pszResponse* // Buffer containing response text
);

Parameters

hSLC

[0115] [in] Handle to current SL client context handle

pszProdKeyHash

[0116] string of the product key hash

pszChallenge

[0117] [in] Response text

Returns

[0118] Success or failure.

SLEnumLicense

[0119] The SLEnumLicensefunction is used to enumerate installed licenses and get information from the license.

HRESULT SLEnumLicense(
HSLC *hSLC,* // SL client context handle
CONST GUID* *pguidApp,* // application GUID
DWORD *dwIndex* // index number
);

Parameters

hSLC

[0120] [in] Handle to current SL client context handle

pguidApp

[0121] [in] See SLInstall. If *pguidApp* is not NULL, then licenses associated with this GUID are enumerated. If the GUID is NULL, then all licenses are enumerated.

dwIndex

[0122] [in] The index number of the license to be retrieved. This value should be zero for the first call to the SLEnumLicense function and then incremented for subsequent calls.

[0123] The function may return licenses in any order.

Returns

[0124] E_SL_NO_MORE_DATA – No license at the specified index.

Remarks

[0125] If SLEnumLicenses succeeded, the selected license information can be accessed by calling SLGetInformation and the category is SL_CAT_ENUMLICINFO

Sample:

```
DWORD i=0;

PBYTE pbProductPriority = NULL;
PBYTE pbRemainingGracePeriodDays = NULL;

for (i=0; ; i++)
{
  EXIT_ON_ERROR(SLEnumLicense(hSLC, NULL, dwIndex));

  if (E_SL_NO_MORE_DATA == SCODE(hr))
  {
    hr = S_OK;
    break;
```

```
    }

    EXIT_ON_ERROR(SLGetInformation(hSLC, SL_CAT_ENUMLIC, "ProductPriority",
&dwType, &cbProductPriority, &pbProductPriority));

    EXIT_ON_ERROR(SLGetInformation(hSLC, SL_CAT_ENUMLIC,
"RemainingGracePeriodDays", &dwType, &cbRemainingGracePeriodDays,
&pbRemainingGracePeriodDays));

Exit:
  SLFreeMemory(pbProductPriority);
  SLFreeMemory(pbRemainingGracePeriodDays);

}
```

SLFreeMemory

[0126] The SLFreeMemory function is used to free the memory allocated by SLC.

```
VOID SLFreeMemory(
  PVOID          pvMemblock,      // pointer to memory
);
```

Parameters

pvMemBlock

[0127] [in] Previously allocated memory block to be freed

Returns

[0128] None

Use of Software Licensing API to Control Use of Software

[0129] A software product uses the API of the present invention for various purposes related to licensing, including the consumption of rights in a license, and the retrieval of data from the license. As noted above, the API allows the software to determine what rights are present in the license, but, preferably, it is up to the software to determine what to do with that information – e.g., grant or deny access to a feature, cease operation altogether, etc. The following description of FIGS. 4 and 5 show how the API of the present invention is used by a software product.

[0130] FIG. 4 shows an example process by which an application "consumes" a right. The application calls the SLConsumeRight method (402). As discussed above, the arguments to the SLConsumeRight function include the client handle assigned by the licensing service, and the name of the right (which is assigned by the vendor of the software to which the right pertains). The licensing service (service 202, shown in FIG. 2) receives the call (404). The service then locates licenses that contains the right, and checks the licenses bindings and validity. As noted above, the license is located in the license store; if there is more than one license that pertains to the application software to which the SLConsumeRight call pertains, then a priority rule may be used to select one of the applicable licenses. Checking the binding means determining that: (1) the license is bound to the product key of the application identified by the client handle; and (2) the license is bound to the machine on which the software is running (or to the group of machines of which the current machine is a member). Checking validity may include determining that the right has not expired (in the case of licenses that specify an expiration date), and that the maximum number of uses of the right is not exceeded (in the case where the license specifies a maximum number of times that the right may be used (i.e., "consumed")).

[0131] If the license and/or right are found to be correctly bound and valid (408), then the license is bound to the right requested in the API call (412). (It should be noted that "binding" a license to a machine, environment, and a product key means that the license specifies which machine(s) and product key it can be used with; "binding" a license to a right means that the consume function has been successful, and the right is being consumed from a particular license. Throughout this description, it will be clear from context which meaning of "binding" applies.) The API call then returns to the calling application and indicates that the call was successful (414). If the license and/or right has been found to be invalid, or not correctly bound to the machine,

environment, or product ID, then the SLConsumeRight call returns to the calling application and indicates that the operation failed (410).

[0132] If the SLConsumeRight call returns with a failure, then the right specified in the call cannot be consumed from a license, and no information about that right will be available to the calling application. However, if the right is successfully consumed, then the application can use the binding of the right to the license to get information from the license about the right. For example, a license may contain a general right called "run," which indicates that the application may be run. However, for the "run" right, the license may contain more specific parameters about the usage of the application – e.g., the license may specify whether particular features of an application (e.g., print, edit, save, etc.) should be turned on or off, and may give specific parameters for the use of these features (e.g., the document can be saved only on machines that are running in a particular domain, or the print feature can only be used for thirty days, etc.). The SL API does not require any particular type of information to be associated with a right, but rather provides a mechanism whereby an application vendor can associate any type of information with a right, which can then be retrieved and interpreted by the application.

[0133] Assuming that a right has been successfully consumed as described in FIG. 4, the application may then retrieve the information associated with the right. The process of retrieving this information is described in FIG. 5.

[0134] First, the application calls the SLGetInformation method on the bound right (502). The various types of information that can be retrieved are described above in connection with the description of the SLGetInformation method. The licensing service then receives the call (504). The service retrieves the requested information from the license file that contains the bound right (506). The licensing service then places this information in a buffer (508), and returns to the calling application (510). The calling application then reads the contents of the buffer, and performs whatever actions it deems necessary based on the retrieved information.

[0135] It should be noted that the licensing service may not be aware of the meaning of the information that it is handling as part of an SLGetInformation call. As discussed above, the licensing framework provides a mechanism whereby a software vendor can create rights, and can associate information with the rights. The invention is not limited either to any particular type of information that can be associated with the right. When the information is retrieved from the

license, it is simply passed by the licensing service to the application in a buffer. The application then interprets the retrieved information, decides what actions to be taken based on that information, and uses its own security features to enforce the application's decision. (E.g., if, based on the retrieved information, the application decides to disable the print feature, the application contains the code that actually disables this features and, possibly, code that prevents a hacker from tampering with the disabling of the print feature.)

[0136] It is noted that the foregoing examples have been provided merely for the purpose of explanation and are in no way to be construed as limiting of the present invention. While the invention has been described with reference to various embodiments, it is understood that the words which have been used herein are words of description and illustration, rather than words of limitations. Further, although the invention has been described herein with reference to particular means, materials and embodiments, the invention is not intended to be limited to the particulars disclosed herein; rather, the invention extends to all functionally equivalent structures, methods and uses, such as are within the scope of the appended claims. Those skilled in the art, having the benefit of the teachings of this specification, may effect numerous modifications thereto and changes may be made without departing from the scope and spirit of the invention in its aspects.